Problema 4

Implementação

qr_House_1.m

Esta é a implementação padrão do método de Householder para fatoração QR. Tranforsma a matriz $A \in \mathbb{M}^{m \times n}$ em uma matriz triangular superior $R \in \mathbb{M}^{m \times n}$ e salvamos os refletores usados para obter Q posteriormente. Essa primeira versão de implementação só abrange os casos onde $m \geq n$.

```
% Entradas:
%   A - matriz de entrada (m x n)
% Saídas:
%   U - matriz (m x m) contendo os vetores normais
%   R - matriz (m x n) triangular superior
function [U, R] = qr_House_1(A)
  [m, n] = size(A);

  % Inicializar a matriz U com zeros
  U = zeros(m, m);

  for i = 1 : n
    % Extrair a coluna atual a partir da i-ésima linha até o final
    x = A(i:m, i);

    % Obtém o vetor normal ao hiperplano de reflexão
    if x(1) > 0
      x(1) = x(1) + norm(x, 2);
    else
      x(1) = x(1) - norm(x, 2);
    end

    u = x / norm(x, 2); % Normaliza o vetor
    U(i:m, i) = u;  % Armazena o vetor em U

    % Aplica a transformação de Householder à submatriz de A
    A(i:m, i:n) = A(i:m, i:n) - 2*u*(u'*A(i:m, i:n));
  end

  R = triu(A); % Os valores abaixo da diagonal seriam proximos de 0, trunco-os
end
```

qr_House_2.m

Seja $A \in M^{m \times n}$. Para quaisquer $m, n \in \mathbb{N}$ a função qr_House_2(A) é capaz de realizar parte da fatoração, o que é feito por completo ao construi Q a partir de U.

Se $m > n$: a função obtém o mesmo resultado que qr_GS.

Se $m = n$: temos $Q \in \mathbb{M}^{m \times m-1}$ e $R \in \mathbb{M}^{m-1 \times n}$.

Se $m < n$: temos $Q \in \mathbb{M}^{m \times m}$ e $R \in \mathbb{M}^{m \times n}$, o resultado geral de qr_House_1, mas agora funcionando para esse caso.

```
% Entradas:
%   A - matriz (m x n)
% Saídas:
%   U - matriz (m x k) contendo os vetores normais
%   R - matriz (k x n) triangular superior
function [U,R] = qr_House_2(A)
  [m, n] = size(A);

  % Determina a dimensão correta
  if m == n
    k = m - 1;
  else
    k = min(m,n); % Essa alteração abrange o caso onde m < n
  end

  % Inicializa matrizes
  R = A;
  U = zeros(m, k);

  for i = 1 : k
    x = A(i:m, i);

    % Obtém o vetor normal ao hiperplano de reflexão
    if x(1) > 0
      x(1) = x(1) + norm(x, 2);
    else
      x(1) = x(1) - norm(x, 2);
    end

    u = x / norm(x,2); % Normaliza o vetor
    U(i:m, i) = u;     % Armazena o vetor em U

    % Aplica a transformação de Householder à submatriz de A
    A(i:m, i:n) = A(i:m, i:n) - 2*u*(u'*A(i:m, i:n));
  end

  R = triu(A(1:k, 1:n)); % Para que coincida com a matriz R de qr_GS
end
```

constroi_Q.m

```
% Entradas:
%   U - matriz (m x n) com vetores de Householder
% Saídas:
%   Q - matriz (m x n) ortogonal
%   versao - 1: Q é m x m, 2: Q é m x n
function [Q] = constroi_Q(U, versao)
  % Obtém dimensões de U e inicializa Q
  [m,n] = size(U);

  Q = eye(m, m);

  for i = 1 : n
    u = U(:,i);

    % Aplica a tranformação de Householder pela direita Q*(H - u*u')
    Q = Q - 2 * Q * (u * u');
  end

  % Ajusta a dimensão de Q para ser compatível com R
  if versao == 2
    Q = Q(:,1:n);
  end
end
```

Testes

Testes - $A(4 \times 3)$

```
>> [UHa, RHa] = qr_House_1(A)
UHa =
    0.86603          0          0    0
   -0.28868    0.97325          0    0
   -0.28868    0.22975    0.80756    0
    0.28868    1.0e-16    0.58979    0

RHa =
   -2        -1   -0.5
    0   -2.2361   -3.3541
    0         0   -1.2247
    0         0    0

>> [UKa, RKa] = qr_House_2(A)
UKa =
    0.86603          0          0
   -0.28868    0.97325          0
   -0.28868    0.22975    0.80756
    0.28868    1.0e-16    0.58979

RKa =
   -2        -1   -0.5
    0   -2.2361   -3.3541
    0         0   -1.2247
```

Como pode ser visto no exemplo acima, a segunda versão retornar um "recorte" do que seria retornado pela primeira versão. Nesse cenário, fica bem claro que isso é, na verdade, uma seleção da informação relevante, afinal foram cortadas as colunas ou linhas de zero.

Construindo as matrizes Q correspondentes, vemos que ambas são muito próximas de serem ortogonais, sem diferença significativa.

```
>> QHa = constroi_Q(UHa, 1)
QHa =
   -0.5   -0.67082    0.40825   -0.36515
    0.5   -0.67082    4.4e-16    0.54772
    0.5   -0.22361   -0.40825    -0.7303
   -0.5   -0.22361    -0.8165    0.18257

>> norm(QHa' * QHa - eye(4)) % Teste de ortogonalidade
    1.7e-15

>> QKa = constroi_Q(UKa, 2)
QKa =
   -0.5   -0.67082    0.40825
    0.5   -0.67082    4.4e-16
    0.5   -0.22361   -0.40825
   -0.5   -0.22361    -0.8165

>> norm(QKa' * QKa - eye(3)) % Teste de ortogonalidade
    1.6e-15
```

Ao multiplicar as matrizes, o resultado indifere da função. Obtemos umas matriz quase igual a A, porém com $9.9e-16$ ao invés de 0 em uma das entradas, um erro insignificante que pode ser desconsiderado.

Testes - $B(4 \times 4)$

```
>> [UHb, RHb] = qr_House_1(B)
UHb =
    0.95471          0          0    0
    0.13469    0.88727          0    0
    0.24243    0.13344   -0.88514    0
    0.10775    0.44152    0.46532    1

RHb =
   -19.442   -10.595   -10.904    -18.516
         0   -16.054   -15.726   -0.98476
         0         0    1.9486    -5.8458
         0         0         0   -5.3e-15

>> [UKb, RKb] = qr_House_2(B)
UKb =
    0.95471          0          0
    0.13469    0.88727          0
    0.24243    0.13344   -0.88514
    0.10775    0.44152    0.46532

RKb =
   -19.442   -10.595   -10.904    -18.516
         0   -16.054   -15.726   -0.98476
         0         0    1.9486    -5.8458
```

Testes - $C(2 \times 4)$

```
% [UHc, RHc] = qr_House_1(C) % Erro devido as dimensoes
[UKc, RKc] = qr_House_2(C)
UKc =
    0.88167    0
    0.47186   -1

RKc =
    -3.6056    -1.3868   -2.7735   -3.8829
          0    0.27735    0.5547    2.2188
```